Efficient Direct-Access Ranked Retrieval

Anonymous Author(s)

Abstract

We study the problem of *Direct-Access Ranked Retrieval* (DAR), a fundamental challenge in large-scale information retrieval and data exploration. DAR arises in applications that require offset-based or paged navigation of ranked results, such as search engines and recommendation systems. The objective is to support efficient access to arbitrary rank positions or windows in the result list, according to a ranking function, without enumerating all preceding items. To address this need, we formalize the DAR problem and propose a theoretically efficient algorithm based on geometric arrangements, achieving logarithmic query time. However, this method suffers from exponential space complexity in high dimensions. Therefore, we develop a second class of algorithms based on ε-sampling, which consume a linear space. Since exactly locating the tuple at a specific rank is challenging due to its connection to the range counting problem, we introduce a relaxed variant called *Conformal Set Ranked Retrieval* (CSR), which returns a small subset guaranteed to contain the target tuple. To solve the CSR problem efficiently, we define an intermediate problem, *Stripe Range Retrieval* (SRR), and design a hierarchical sampling data structure tailored for narrow-range queries. Our method achieves practical scalability in both data size and dimensionality. We prove near-optimal bounds on the efficiency of our algorithms and validate their performance through extensive experiments on real and synthetic datasets, demonstrating scalability to millions of tuples and hundreds of dimensions.

1 Introduction

Ranked retrieval is a cornerstone of modern information access, powering interactive web applications, data exploration, and recommendation systems [8, 12, 13, 24, 58]. Results are typically ordered by a *query-specific*, *user-specific*, or *machine-generated* scoring function (aka. ranking function), while users may need to access not only the very top results but also *arbitrary* rank positions or windows (direct-access retrieval). This occurs in common scenarios such as paged navigation of web search results [24], browsing ranked product lists in e-commerce [55], or scrolling through personalized feeds on social platforms [45]. In practice, web applications implement this through paged or offset-based navigation.

Example 1. *In web and e-commerce applications, APIs commonly use* offset-based pagination. *For instance, a request such as [*`GET /items?offset=40&limit=5`*] retrieves items ranked 41–45. More generally, this pattern requires efficient retrieval of results at deeper ranks without scanning all earlier entries [55, 63].*

A major challenge, however, is that *retrieving results at deeper offsets is notoriously inefficient*, as it often requires enumerating all preceding items.

Similar issues also arise in data science and visual analytics tooling, where analysts frequently interact with large, ranked datasets. Tools such as spreadsheets, notebooks, and visual analytics platforms provide intuitive interfaces for exploration [22, 56, 60, 71], but the explosive growth of data volumes driven by big data and AI has placed heavy demands on their efficiency and responsiveness [10].

Example 2. *In spreadsheet applications [46, 57], data scientists frequently create new attributes as weighted sums of existing ones. Subsequently, they may* "jump" *to tuples at specific rank positions without explicitly enumerating preceding entries.*

More examples can be found for other applications such as Jupyter notebooks [43], where analysts may need to retrieve tuples at particular ranks, effectively *skipping* over earlier-ranked items.

Whether in web applications or scientific workflows, the ability to perform *efficient direct-access ranked retrieval*, i.e., to quickly access the tuple[1] at an arbitrary rank position i, without the overhead of enumerating the entire dataset up to that position, is crucial for maintaining the *interactivity* and *responsiveness* expected by users. Particularly, the large number of features (aka. attributes) in domains such as AI create a *high-dimensional space of possible ranking functions*, potentially encompassing up to tens or hundreds of dimensions, especially when ranking functions are machine-generated [45, 50].

Traditionally, the research focus in ranked retrieval has been on *top-k* query processing, with the objective of finding the *top-ranked* tuples [40]. Such approaches, however, are tailored for a small value of k, and are ineffective for direct-access queries that have the goal of efficiently retrieving the tuple at an arbitrary ranking position i, where i can be in $O(n)$. Direct access to conjunctive queries has recently been studied to directly access a specific position in the join result, without fully materializing the join [15, 31, 66]. These works, however, consider an *prespecified ordering* of the tuples (e.g., lexicographic order on an attribute), and focus on skipping the join operation. As a result, such approaches are not suitable for the settings where the ranking function is part of the input query.

Therefore, in this paper, we formalize and study the *direct-access ranked retrieval* (DAR) problem. We also introduce the notion of *conformal set* for ranked retrieval as a small set that contains the tuple at the queried rank i. Next, focusing on linear ranking functions, we propose efficient algorithms for each of the proposed problems. Specifically, our first algorithm uses the computational geometric concepts of duality, arrangements, and levels of arrangements for answering the DAR problem. This algorithm, however, suffers from the curse of dimensionality, and its space complexity increases exponentially with the number of attributes.

Consequently, we propose several algorithms with a linear space complexity, even for a large number of attributes. Specifically, starting from $d = 2$, we propose our algorithm based on the concept of ε-samples, and extend it to high-dimensional settings by combining it with the stripe-range searching. This approach offers an efficient solution for finding a conformal set for an arbitrary rank position. Finally, combining our approach with range counting, we extend it for the exact version (DAR problem).

Efficiently finding the tuples within a *narrow* stripe range is fundamental in the development of our ε-sampling based solution. We formulate this as the problem of *stripe range* retrieval (SRR),

[1]We use the terms *tuple* and *item* interchangeably.

with a focus on developing efficient algorithms for the challenging narrow ranges that only contain a small number of tuples.

Inspired by the Hierarchical Navigable Small World graphs for approximate nearest neighbor problems [27, 47], our practical and scalable algorithm constructs a hierarchical sampling structure for stripe range searching, maintaining the neighborhood regions of points in the hierarchy using smallest enclosing balls.

Theoretically, we show the near-optimal efficiency of our algorithms up to a logarithmic factor. Besides, we conduct extensive experiments on real-world and synthetic datasets to evaluate the performance of our algorithms in practice and compare them against the existing baselines. Our experiments confirm the *efficiency and scalability* of our proposed algorithms. In particular, the ε-sampling algorithm, when combined with the hierarchical structure, achieves the best scalability with respect to both dimension and dataset size in terms of query time and index size, whereas the baseline methods are mostly affected by the curse of dimensionality.

Summary of Contributions.

- (Section 2) We introduce and formalize the problems of (i) direct-access ranked retrieval (DAR), (ii) conformal set ranked retrieval (CSR), and (iii) stripe range retrieval (SRR).
- (Section 4) We provide a lower bound on the DAR problem as well as proving the near-optimality of our algorithms up to a logarithmic factor.
- (Appendix A) We propose an algorithm based on the geometric concepts of duality and the arrangement of hyperplanes that offers a logarithmic query time, suitable for low-dimensional settings, especially when $d = 2$.
- (Section 5) We propose several algorithms with linear space usage based on ε-sampling combined with range searching to offer efficient solutions for the CSR and DAR problems.
- (Section 5.3) We propose a practical solution for the stripe range retrieval problem (SRR), based on hierarchical random sampling and keeping track of the proximity of points. This solution also enables a practical algorithm for the CSR problem, scalable to large-scale settings, where we have to deal with narrow stripe ranges.
- (Section 6) We conduct comprehensive experiments to demonstrate the practical performance of our algorithms across different settings on synthetic and real-world scenarios.
- (Appendix C) We extend our structures to the dynamic setting, supporting `Insert` and `Delete` operations.

In addition, the background concepts relevant to our algorithms are reviewed in Section 3, the related work is discussed in Section 7, and the final remarks are provided in Section 8.

2 Formal Definitions

We consider a dataset $\mathcal{D} = \{p_1, p_2, \cdots, p_n\}$ consisting of n points in $\mathbb{R}^d$, where each point $p \in \mathcal{D}$ has d real-valued (scoring) attributes, aka. features, used for ranking.[2] A *scoring function* $\texttt{score} : \mathbb{R}^d \to \mathbb{R}$ assigns a real-valued score to each point in the dataset.[3] In this work, we focus on *linear scoring functions*, where the score is defined as a weighted sum of the features. Specifically, given a weight vector $f \in \mathbb{R}^d$, the score assigned to a point p is computed as the dot product[4]: $\texttt{score}_f(p) = f^\top p = \sum_{j=1}^{d} f[j]p[j]$. We may directly call f as the scoring function. Figure 1 (left) provides an example of a dataset $\mathcal{D}$ with $n = 16$ points and two attributes x and y. Let the red origin-anchored ray in the figure represent f. The perpendicular projection of each point p on the vector f shows $\texttt{score}_f(p)$.

[2]Besides scoring attributes, the dataset may contain other attributes such as non-ordinal descriptive attributes that are not used for ranking. In this paper, we use the terms attribute and feature interchangeably to refer to the scoring attributes.
[3]We use the terms scoring function and ranking function interchangeably.

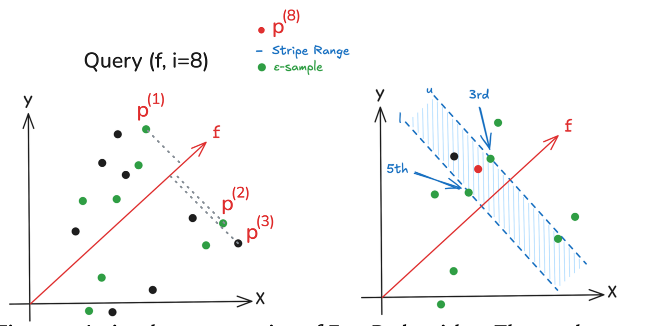

Figure 1: A visual representation of EPS2D algorithm. The toy dataset contain 16 points and the query asks for $(i = 8)$-th point with the scoring function shown in red. (Left) First, we find the bounds by solving DAR on the ε-sample. (Right) Finding the ℓ and u values, we solve the SRR problem on the stripe range.

2.1 Direct-Access Ranked Retrieval (DAR)

Given a scoring function f, we define a total order over the dataset $\mathcal{D}$ by sorting the points in descending order of their scores. For example, in Figure 1, the ordering of the projections on the vector of f shows their ranking. Let $\texttt{rank}_{\mathcal{D},f} : \mathcal{D} \to \mathbb{N}$ denote the function that assigns to each point in $\mathcal{D}$ its rank in this ordering (i.e., $\texttt{rank}_{\mathcal{D},f}(p) = i$ if p is the i-th highest-scoring point under $\texttt{score}_f$). When the dataset $\mathcal{D}$ is clear from the context, we write $\texttt{rank}_f$ for brevity. We denote by $p^{(i)}$ the i-th point (aka the i-th element) under $\texttt{score}_f$, that is, $p^{(i)} = \texttt{rank}_f^{-1}(i), \quad \forall\, 1 \le i \le n$.

Our main problem in this paper is to efficiently find the point at a specific rank i.

Problem 1 (Direct-Access Ranked Retrieval (DAR)). *Given a dataset $\mathcal{D}$ available at preprocessing time, a query consists of a linear scoring function $f \in \mathbb{R}^d$ and an integer $i \in [n]$, find the point $p^{(i)}$, with rank i in $\mathcal{D}$ according to the scoring function $\texttt{score}_f$. i.e. return:* $p^{(i)} = \texttt{rank}_{\mathcal{D},f}^{-1}(i)$.

As proved in Appendix G of the technical report [64], finding $p^{(i)}$ in high-dimensional settings is challenging since its complexity is tied to the complexity of *half-space range counting* problem [1, 18].

Therefore, inspired by the machine learning concept of conformal predictions [5, 59], we introduce a "relaxed" version of Problem 1, where instead of returning one point $p^{(i)}$, we return *a small set that contains* $p^{(i)}$ – referred as the *conformal set* for rank i.

Problem 2 (Conformal-Set Ranked Retrieval (CSR)). *Given a dataset $\mathcal{D}$ available at preprocessing time, a query consists of a linear scoring function $f \in \mathbb{R}^d$, an integer $i \in [n]$, and a small value $\kappa \ll n$, find a conformal set $C(i) \subset \mathcal{D}$, where $|C(i)| \le \kappa$ and $\texttt{rank}_{\mathcal{D},f}^{-1}(i) \in C(i)$.*

[4]We assume that the scoring vector f is normalized, i.e., $\|f\| = 1$.

2.2 Stripe Range Retrieval (SRR)

While addressing our main problems (Problem 1 and Problem 2), we also address a side problem that finds all points that fall in a stripe range. The solution to this problem helps us in developing efficient solutions for our main problems. Formally, we define a stripe range as follows.

Stripe Range. A *stripe range* $\mathcal{S}_{f,\ell,u}$ is defined by a scoring function $f \in \mathbb{R}^d$ and two real-valued boundaries $\ell, u \in \mathbb{R}$ such that $\ell \leq u$. It corresponds to the universe of all points $x \in \mathbb{R}^d$ whose scores based on f lies within the interval $[\ell, u]$, i.e.,

$$\mathcal{S}_{f,\ell,u} = \{x \in \mathbb{R}^d \mid \ell \leq \mathtt{score}_f(x) \leq u\}.$$

Given a stripe range query, we aim to return the set of points in $\mathcal{D}$ that fall inside the given stripe range. For example, in Figure 1, the stripe range query is highlighted between the two blue lines, and the points within this query should be returned.

Problem 3 (Stripe Range Retrieval (SRR)). *Given a dataset $\mathcal{D}$ and a query consisting of a* stripe range $\mathcal{S}_{f,\ell,u}$*, return all points in $\mathcal{D}$ that lie within this range, i.e., $\mathcal{D}_o = \mathcal{S}_{f,\ell,u} \cap \mathcal{D}$.*

3 Background

In this section, we summarize the key concepts and definitions used in the subsequent discussions. We assume a range space $(\mathcal{D}, \mathcal{R})$, where $\mathcal{D}$ is the dataset, and $\mathcal{R}$ is a collection of geometric ranges defined over $\mathcal{D}$, such as all possible balls, half-spaces, or other regions in $\mathbb{R}^d$.

VC-Dimension. The Vapnik-Chervonenkis (VC) dimension [68] for a range space is the size of the largest subset $C \subseteq \mathcal{D}$ such that every subset of C can be realized as the intersection of C with some range in $\mathcal{R}$. A low VC-dimension implies a limited expressive power, which in turn allows for efficient sampling-based approximations [37].

ε-Samples. An ε-sample of a range space $(\mathcal{D}, \mathcal{R})$ is a subset $\mathcal{N} \subseteq \mathcal{D}$ such that, for every range $R \in \mathcal{R}$, the proportion of points in R is approximately preserved in $\mathcal{N}$. More formally, $\mathcal{N}$ is an ε-sample if for all $R \in \mathcal{R}$,

$$\left| \frac{|R \cap \mathcal{D}|}{|\mathcal{D}|} - \frac{|R \cap \mathcal{N}|}{|\mathcal{N}|} \right| \leq \varepsilon.$$

For more details on ε-sampling, we refer the reader to [37]. We use the following foundational theorem on ε-samples as a key tool in deriving our theoretical guarantees:

Theorem 1 (ε-sample theorem [68]). *Let $(\mathcal{D}, \mathcal{R})$ be a range space of VC-dimension δ. Then for any $\varepsilon, \varphi \in (0,1)$, a random sample $S \subseteq \mathcal{D}$ of size $O\left(\frac{\delta}{\varepsilon^2}\log\frac{\delta}{\varepsilon} + \frac{\delta}{\varepsilon^2}\log\frac{\delta}{\varphi}\right)$ is an ε-sample of $(\mathcal{D}, \mathcal{R})$ with probability at least $1 - \varphi$.* □

ε-Sample Construction. In addition to random sampling, which provides a probabilistic method for constructing ε-samples, there also exist deterministic approaches based on discrepancy theory [19, 37]. These methods can achieve near-linear runtime.

4 Solution Overview

In this section, we provide a high-level overview of our proposed algorithms, with a summary of their theoretical guarantees.

The baseline approach for answering the DAR queries uses the extension of the classic divide and conquer (D&C) median finding algorithm, which, instead of the median, finds the element at position i [11, 23]. To do so, it first makes a linear pass over $\mathcal{D}$, and for each point p, computes $\mathtt{score}_f(p)$. Then, using the D&C algorithm, it finds $\mathtt{rank}_f^{-1}(i)$ in $O(n)$. Instead, our algorithms, summarized in Table 1, provide more efficient solutions by preprocessing the data and constructing proper data structures:

KthLevel. Our first algorithm for solving the DAR problem using the computational geometric concepts of *duality* and *arrangement* of hyperplanes [21, 30]. At a high level, building the arrangement of dual hyperplanes of the tuples, our approach, named KthLevel, keeps track of different *levels of the arrangement*, which are used at the query time for efficient answering of DAR queries, i.e., finding the tuple at a given rank position i. Due to the space constraints, further details about this algorithm and its analysis are presented in Appendix A.

This method is an exact solution to the DAR problem that offers highly efficient *logarithmic query time*, but it is not practical for high-dimensional settings ($d > 2$) as it has a *space complexity exponential to the number of attributes* d. Our subsequent algorithms based on ε-samples aim at maintaining a linear space complexity.

ε-sampling. We start by the simple two-dimensional case. During the preprocessing, we compute an ε-sample of the dataset, denoted by $\mathcal{N}_\varepsilon$. Given a scoring function f and a target rank i, we approximately solve the DAR problem on the smaller set $\mathcal{N}_\varepsilon$, rather than the full dataset $\mathcal{D}$. From this approximate solution, we derive a stripe range containing the true i-th ranked point. By solving an instance of stripe range searching (SRR) we find this exact point. We refer to this algorithm as Eps2D. A visual representation of this approach is shown in Figure 1 (left and right).

For higher-dimensional settings, we follow a similar generalized strategy (EpsRange). However, unlike in 2D, standard range searching algorithms do not perform well in high-dimensional datasets. To overcome this, we propose a practical algorithm to solve the intermediate SRR problem with hierarchical sampling (EpsHier).

Lower Bounds and Optimality. A discussion on the lower bounds and optimality of our algorithms is provided in the Appendix G of the technical report [64].

5 ε-sampling Approach

In theory, the solution based on constructing the arrangement of dual hyperplanes provides a logarithmic time for answering the DAR queries. It, however, is not practical especially when the number of ranking attributes is not small (≥ 3), since the size of the arrangement (hence, the space complexity) increases exponentially with d. Therefore, in this section, we propose practical solutions that keep the space complexity linear, i.e., $O(n)$. Our algorithms are based on the computational geometric concept of ε-samples [37] (also introduced in Section 3).

In the following, we start by discussing the two-dimensional Eps2D algorithm, then, we will generalize our solution to the higher dimensions, and propose practical methods to solve the intermediate SRR problem (Problem 3).

Algorithm Name	Summary	Dimension	Space Complexity	Query Time	Problem
KthLevel	Following the levels of arrangement	$d \geq 2$	$O(n^d)$	$O(\log n)$	DAR
Eps2D	ε-sampling in 2D	$d = 2$	$O(n)$	$O(n^{2/3} \log n)$	DAR
EpsRange	ε-sampling + Range Searching	$d \geq 2$	$O(n)$	$O(\max\{n^{1-1/d}, \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon}\})$	CSR
				$O(\max\{n^{1-1/d}, n^{2/3} \log n\})$	DAR
EpsHier	ε-sampling + Hierarchical Sampling	$d \geq 2$	$O(n)$	Worst-case $O(n)$; Practically Efficient	CSR

Table 1: Overview of the proposed methods for the DAR problem.[5]

5.1 Eps2D: ε-sampling in 2D

In this section, we discuss our solution based on ε-samples in 2D, called Eps2D.

Preprocessing. During the preprocessing phase, the algorithm computes an ε-sample of the dataset $\mathcal{D}$, where the ranges are *stripe ranges*. We denote this ε-sample by $\mathcal{N}_\varepsilon$, for a given parameter ε (see Section 3 for background).

Query Answering. Let $m = |\mathcal{N}_\varepsilon|$ denote the size of the ε-sample build during the preprocessing time. Note that $m \ll n$.[6]

(Step i: Thresholding) During the *query phase*, given a query pair (f, i), we begin by solving the DAR problem on the ε-sample $\mathcal{N}_\varepsilon$. Specifically, we identify the i_ℓ-th and i_u-th ranked points in $\mathcal{N}_\varepsilon$ with respect to the scoring function f, where:

$$i_\ell = \left\lfloor m \left(\frac{i}{n} - \varepsilon \right) \right\rfloor, \quad i_u = \left\lceil m \left(\frac{i}{n} + \varepsilon \right) \right\rceil, \tag{1}$$

and $m = |\mathcal{N}_\varepsilon|$. This step can be done by sorting the points in $\mathcal{N}_\varepsilon$ according to $\texttt{score}_f$ or it can be done with linear time in m by using the baseline median finding algorithms, like median of medians. These two points serve as score-based thresholds that are guaranteed to bound the score of the i-th ranked point in the full dataset $\mathcal{D}$, as stated in the following lemma.

Lemma 2. *Let $\mathcal{N}_\varepsilon$ be an ε-sample of the dataset $\mathcal{D}$ for some $\varepsilon > 0$. Then, for any linear scoring function f, the score of the i-th ranked point $p^{(i)}$ in $\mathcal{D}$ is bounded as*

$$\texttt{score}_f(q_\ell) \leq \texttt{score}_f(p^{(i)}) \leq \texttt{score}_f(q_u),$$

where q_ℓ and q_u are the i_ℓ-th and i_u-th ranked points in $\mathcal{N}_\varepsilon$ under f, i.e.,

$$q_j = \texttt{rank}^{-1}_{\mathcal{N}_\varepsilon, f}(i_j), \quad \textit{for } j \in \{\ell, u\}.$$

□

Proof. This comes from the definition of ε-samples. See Appendix B for proof. □

(Step ii: SRR) Next, we compute the score thresholds corresponding to the boundary points identified in the previous step:

$$\ell = \texttt{score}_f(q_\ell), \quad u = \texttt{score}_f(q_u). \tag{2}$$

Using these values, we define the stripe range $\mathcal{S}_{f,\ell,u}$. We then solve an instance of the SRR problem on the dataset $\mathcal{D}$ using this stripe range. The result is a subset of points within the score interval $[\ell, u]$, which we denote by $\mathcal{D}_o$. This set contains the i-th ranked point $p^{(i)}$, since $\mathcal{N}_\varepsilon$ is an ε-sample. Note that this set is a conformal set for rank i, hence providing a solution to the CSR problem (Problem 2).

[5]The Big-Oh notations in our guarantees hide constants that are polynomially dependent on the dimension d. Consistent with the prior literature [1], our focus is on the exponential dependence on d.

[6]See Theorem 1, while noting the VC-dimension of stripe ranges is $\delta = d + 1$.

Algorithm 1 Eps2D-Query($\mathcal{N}_\varepsilon, \mathcal{D}, f, i$)

```
Input: Precomputed ε-sample $\mathcal{N}_\varepsilon$ of size $m$, original dataset $\mathcal{D}$, scoring vector $f$, target rank $i$
Output: The exact $k$-th ranked point $p^{(i)}$ in $\mathcal{D}$ under $\texttt{score}_f$
1: Calculate $i_\ell$ and $i_u$                                  ▷ equation 1
2: $q_\ell \leftarrow \texttt{rank}^{-1}_{\mathcal{N}_\varepsilon,f}(i_\ell)$,  $q_u \leftarrow \texttt{rank}^{-1}_{\mathcal{N}_\varepsilon,f}(i_u)$        ▷ Median of medians algorithm
3: $\ell \leftarrow \texttt{score}_f(q_\ell)$,  $u \leftarrow \texttt{score}_f(q_u)$        ▷ end of step i
4: $\mathcal{D}_o \leftarrow \mathcal{S}_{f,\ell,u} \cap \mathcal{D}$                     ▷ via SRS query; step ii
5: $|H_u| \leftarrow \left|\{p \in \mathcal{D} \mid f^\top p \geq u\}\right|$     ▷ via range counting; step iii
6: Sort $\mathcal{D}_o$ ascending by $\texttt{score}_f$
7: return the $(i - |H_u|)$-th point in the sorted list          ▷ step iv
```

(Step iii: Range Counting) We define the half-space H_u as,

$$H_u = \{p \in \mathcal{D} \mid p^\top f \geq u\}, \tag{3}$$

H_u contains all points in $\mathcal{D}$ whose score under f is at least u. We then apply a half-space range counting algorithm in two dimensions to compute the number of points lying within H_u [48].

(Step iv: Final Selection) In the final step, we sort the points in the candidate set $\mathcal{D}_o$ in descending order according to the scoring function $\texttt{score}_f$. Let $|H_u|$ denote the number of points in $\mathcal{D}$ whose score is greater than or equal to u, as computed in the previous step. We then return the $(i - |H_u|)$-th point in the sorted list as the exact answer to the DAR query.

A pseudo-code of Eps2D is shown in Algorithm 1.

5.1.1 Analysis. We analyze the time and space complexity of the Eps2D algorithm step-by-step. (Step i) Finding the i_ℓ-th and i_u-th point in $\mathcal{N}_\varepsilon$ of size m takes $O(m)$ using the extension of the median finding algorithm. (Step ii) This step involves solving a simplex range searching query in 2D (with stripe ranges). Using the Matoušek's efficient range searching technique [48], this can be done in $O(\sqrt{n} + |\mathcal{D}_o|)$ time with $O(n)$ space. (Step iii) We perform a half-space range counting query. Again using Matoušek's partition tree [48], this takes $O(\sqrt{n})$ time and $O(n)$ space. (Step iv) Finally, we sort the set $\mathcal{D}_o$, which requires $O(|\mathcal{D}_o| \log |\mathcal{D}_o|)$ time.

Space Complexity. All operations use data structures with linear space requirements. Thus, the total space usage is linear in input.

Query Time Complexity. To analyze the runtime precisely, we bound the size of the result set $\mathcal{D}_o$ as follows:

Lemma 3. *The size of the result set $\mathcal{D}_o$ is $O(\varepsilon n)$.* □

Proof. See Appendix B for proof. □

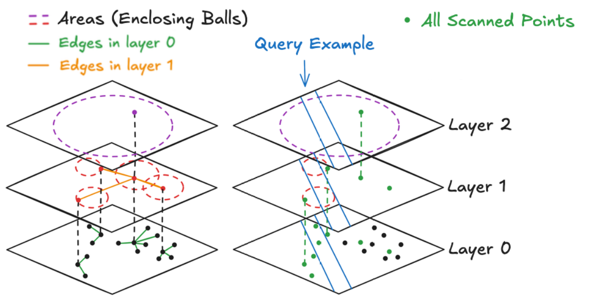

Figure 2: A visual representations of the Hierarchical Sampling structure for solving SRR problem. In this setting, decay rate $r = 4$. (Left) The data structure built during the preprocessing. (Right) An example of running a query on top of this structure, the green points are all the points scanned during the query phase.

In 2D, the size of the ε-sample satisfies (see Section 3): $m = O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)$. Combining all steps, the total runtime is:

$$T(\text{Eps2D}) = O\left(m + \sqrt{n} + |\mathcal{D}_o|\log|\mathcal{D}_o|\right).$$

Using $|\mathcal{D}_o| = O(\varepsilon n)$ and choosing $\varepsilon = n^{-1/3}$ (which minimizes the overall runtime), we get:

$$T(\text{Eps2D}) = O\left(n^{2/3}\log n\right).$$

Theorem 4. *The Eps2D algorithm solves the exact DAR problem in 2D using linear space and $O(n^{2/3}\log n)$ query time.*[7]

5.2 EpsRange: ε-sampling in Higher Dimensions

Next, we present the generalization of the Eps2D algorithm to higher dimensions ($d \geq 2$), referred to as the EpsRange algorithm. The preprocessing phase remains unchanged: we compute an ε-sample $\mathcal{N}_\varepsilon$ of the dataset $\mathcal{D}$, of size m, for a chosen parameter ε. We know that $m \ll n$, because the VC-dimension of stripe ranges is $\delta = d + 1$ [37].

During query time, the algorithm proceeds similarly to Eps2D. In Step i, we find the i_ℓ-th and i_u-th points in $\mathcal{N}_\varepsilon$ according to the scoring function f. In Step ii, we solve a high-dimensional instance of the SRR problem. The resulting set $\mathcal{D}_o$ contains candidate points from $\mathcal{D}$ that lie between the scores of the i_ℓ-th and i_u-th ranked points in the sample. This set is a conformal set for Problem 2. To obtain the exact solution, we proceed to Step iii, where we perform a high-dimensional range counting query to determine the exact rank of each point in $\mathcal{D}_o$. Finally, in Step iv, we sort the candidates in $\mathcal{D}_o$ and return the point with exact rank i, completing the solution to the DAR selection problem.

Analysis. The analysis is similar to the 2D case, provided in Appendix D. The results are as follows:

Theorem 5. *The EpsRange algorithm solves the relaxed version of DAR problem with linear space usage and $O(\max\{n^{1-1/d}, \frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}\})$ time. The exact version can also be solved with the same space usage and $O(\max\{n^{1-1/d}, dn^{2/3}\log n\})$ time.*

5.3 Stripe Range Searching with Hierarchical Sampling

As observed in the previous section, the main bottleneck in the runtime of the EpsRange algorithm lies in the range searching subroutines used to solve the SRR problem. These subroutines often suffer from the curse of dimensionality, particularly for general simplices, such as non-orthogonal half-spaces within stripe ranges. Even algorithms with theoretical sublinear time guarantees tend to perform poorly in practice [49], with a runtime comparable to or worse than a naive linear scan over the entire dataset (see experiments in Section 6).

To address this limitation, we propose a practical algorithm for solving the SRR problem, where the ranges are stripes in d dimensions. We empirically show that augmenting EpsRange with this solution leads to significant performance improvements on real-world high-dimensional datasets (called EpsHier algorithm).

We present a hierarchical data structure for solving the stripe range searching problem. We begin with a high-level overview of the structure, followed by a description of the *preprocessing step*, and conclude with the details of the *query phase* of the algorithm.

Overview. We observe that the stripe range queries constructed in the solution of the DAR problem correspond to *narrow regions, especially for small choices of ε*. To exploit this, we construct a hierarchical structure over the input point set that enables efficient pruning of irrelevant points when answering such range queries.

The core idea is to organize the points in a hierarchy that preserves spatial proximity, inspired by similar structures used in nearest neighbor search [27, 38, 47, 53].[8] Each point in the hierarchy maintains a list of neighbors, and during preprocessing, we compute and store an *enclosing ball* around each such neighborhood.[9]

During query processing, we employ a simple yet effective heuristic: if the stripe intersects the enclosing ball of a node, we explore the node; otherwise, we safely prune it. This significantly reduces the number of points examined during query time.

Empirical results demonstrate that for the narrow stripe ranges used in the DAR problem, this approach leads to a fast algorithm by eliminating many unnecessary explorations. See Figure 2 (right), which illustrates the points explored during an example query.

5.3.1 *Preprocessing.*

Layers. Given the input dataset $\mathcal{D}$, we construct a hierarchical graph $\mathcal{G}(\mathcal{D})$ on top of it. The graph consists of L layers built via recursive random sampling. We define the base layer as the dataset itself, $\mathcal{L}_0 = \mathcal{D}$. Each subsequent layer $\mathcal{L}_\ell$ is formed by randomly sampling $\frac{|\mathcal{L}_{\ell-1}|}{r}$ points from the previous layer $\mathcal{L}_{\ell-1}$, where r is a hyperparameter known as the *exponential decay rate*. Thus, the size of each layer satisfies:

$$|\mathcal{L}_\ell| = \frac{|\mathcal{L}_{\ell-1}|}{r}, \quad \forall\, 1 \leq \ell \leq L$$

Edges. Once the layer $\mathcal{L}_\ell$ is sampled, each point in $\mathcal{L}_\ell$ serves as a centroid for partitioning the points in the layer below, $\mathcal{L}_{\ell-1}$. Specifically, each point $p \in \mathcal{L}_{\ell-1}$ is connected to its nearest neighbor in

[7]Note that we assumed the set $\mathcal{N}$ to be an ε-sample. According to Theorem 1, this happens with high probability via random sampling.
[8]See Related Work (section 7) for comparison.
[9]Not necessarily centered at that point.

$\mathcal{L}_\ell$, forming a *directed edge* from the centroid c^* to p, where

$$c^* = \arg\min_{c \in \mathcal{L}_\ell} ||p - c||_2. \quad (4)$$

This procedure creates a layered structure, where each level induces a graph by connecting lower-layer points to their nearest centroids in the current layer (see Figure 2, left, where each level illustrates the induced graph). For every node p in the graph $\mathcal{G}(\mathcal{D})$, let $N_\ell(p)$ denote the set of its neighbors at layer $\mathcal{L}_\ell$. For example, in Figure 2 (left), the root node has three red neighbors at layer 1 and five black neighbors at layer 0.

Area of Nodes. For each node p in layer ℓ of the graph $\mathcal{G}(\mathcal{D})$, we associate a set called the *area* of the node, denoted by $\mathcal{A}_\ell(p)$. The area captures the set of points from the base layer that are hierarchically covered by this node. The definition is recursive. For the base layer, we set: $\mathcal{A}_0(p) = \{p\}$. For any node p in layer $\ell > 0$, the area is defined as:

$$\mathcal{A}_\ell(p) = \bigcup_{q \in N_{\ell-1}(p)} \mathcal{A}_{\ell-1}(q),$$

where $N_{\ell-1}(p)$ denotes the neighbors (i.e., children) of p in layer $\ell - 1$. In other words, the area of a node at layer ℓ is the union of the areas of all its children in the layer below. As illustrated in Figure 2, we represent the area of each node using the smallest enclosing circle, shown with dotted lines.

Enclosing Balls. For each node p at layer ℓ, we define $\mathcal{B}_\ell(p)$ as the *smallest enclosing ball* that covers the area $\mathcal{A}_\ell(p)$. These balls are later used during query time to enable efficient pruning of nodes when searching within a stripe range:

$$\mathcal{B}_\ell(p) = \text{Enclosed-Ball}(\mathcal{A}_\ell(p)).$$

Construction Algorithm. To construct the hierarchical graph $\mathcal{G}(\mathcal{D})$, we employ a bottom-up recursive approach. Starting from the base layer, we iteratively sample a subset of points to form the next layer. At each level, we connect nodes in the current layer to their nearest centroids in the layer above, thereby forming directed edges. After establishing the connections, we compute and update the area sets and corresponding enclosing balls for the newly constructed layer. The pseudo-code of the preprocessing procedure is detailed in Algorithm 4 in Appendix F.

Analysis. See Appendix E for the analysis of this algorithm.

Querying Algorithm. Given a stripe range $\mathcal{S}_{f,\ell,u}$, the goal is to report all points in $\mathcal{S}_{f,\ell,u} \cap \mathcal{D}$. The query process begins at the top of the hierarchy, starting from layer $\mathcal{L}_L$. At each layer ℓ, we examine whether the enclosing ball $\mathcal{B}_\ell(p)$ of each point $p \in \mathcal{L}_\ell$ intersects the stripe $\mathcal{S}_{f,\ell,u}$. If there is no intersection, the corresponding node is pruned from further exploration. Otherwise, we proceed to explore its neighbors in the next lower layer, denoted by $N_{\ell-1}(p)$. The full query procedure is provided in Algorithm 2. A visual illustration of this process is shown in Figure 2 (right).

6 Experiments

Our code and technical report are available on this anonymous repository. Appendix H of the technical report [64] contains more details on experiments.

Datasets. We use synthetic and real-world datasets. The synthetic dataset is generated from a Zipfian distribution. Our real-world datasets include US Used Cars [52] (3M records, 66 columns), FIFA 2023 [44] (300K records, 54 columns), and US Flights [67] (5M records, 31 columns). Additional details are provided in Appendix H of the technical report [64].

Algorithm 2 Hierarchical-Sampling-Query$(\mathcal{G}(\mathcal{D}), \mathcal{S}_{f,\ell,u})$

Input: The preprocessed graph $\mathcal{G}(\mathcal{D})$ and the query stripe range $\mathcal{S}_{f,\ell,u}$.
Output: The set of points $\mathcal{D}_o = \mathcal{S}_{f,\ell,u} \cap \mathcal{D}$.

```
1:  $\mathcal{D}_o \leftarrow \emptyset$                                  ▷ Placeholder for result
2:  $candidates \leftarrow \mathcal{L}_L$
3:  for $curr\_layer$ from $L$ to 1 do                             ▷ top-down
4:      $tmp \leftarrow \emptyset$                          ▷ Placeholder for new candidates
5:      for each point $p \in Candidates$ do
6:          if $\mathcal{B}_{curr\_layer}(p) \cap \mathcal{S}_{f,\ell,u} \neq \emptyset$ then
7:              $tmp \leftarrow tmp \cup N_{curr\_layer}(p)$       ▷ Explore the area
8:      $candidates \leftarrow tmp$
9:  for $p \in candidates$ do
10:     if $p \in \mathcal{S}_{f,\ell,u}$ then
11:         $\mathcal{D}_o \leftarrow \mathcal{D}_o \cup \{p\}$
12: return $\mathcal{D}_o$
```

Baselines and Methods. We evaluate our methods on the SRR, DAR, and CSR problems. For the SRR problem, we compare against classical and widely used range searching indices, including KD-Tree [9], R-Tree [36], and Partition Tree [48], and Ball Trees [54]. Our proposed method (Hierarchical Sampling) is abbreviated as Hierarchical. As a baseline, we also include a simple linear search approach, denoted as Exhaustive. To apply these indices to stripe range queries, we traverse the tree top-down, pruning subtrees whose corresponding hyperrectangles do not intersect the query stripe (similar to Algorithm 2).

For the DAR and CSR problems, we include the Threshold Algorithm (TA) and Fagin's algorithm [33] as baseline methods. Additionally, we consider a simple linear exhaustive baseline, referred to as Exhaustive, which computes the score for all points, sorts them accordingly, and returns the i-th ranked element. Our proposed algorithms are EpsRange and EpsHier, and are used for the CSR problem. We also include KthLevel, the algorithm based on arrangement construction for DAR problem.

We begin by presenting experiments on the intermediate SRR problem (section 6.1). We then proceed to the end-to-end evaluation of the full DAR problem (section 6.2).

6.1 Stripe Range Retrieval

6.1.1 Synthetic Dataset. We begin by evaluating the performance of the algorithms on synthetic datasets.

Effect of Stripe Width. For these experiments, the scoring function f is sampled uniformly from the unit hypersphere in $\mathbb{R}^d$. We vary the stripe $\mathcal{S}_{f,\ell,u}$ by adjusting its width, i.e., the number of points it contains. Figure 3 presents the query time performance across different dimensionalities d as a function of stripe width. We observe that both KD-Tree and R-Tree methods fail to scale effectively to high-dimensional settings, and thus we exclude them from experiments with $d = 128$. The proposed Hierarchical Sampling method achieves **up to a 16× speedup** over Exhaustive search, particularly in narrow stripe queries. Such narrow stripes frequently arise when solving the DAR and CSR problems, indicating that this method is also helpful for DAR problem. While the Partition Tree also shows a fast query time in low-dimensional spaces, its performance degrades with increasing dimensionality. Appendix H.2, in the technical report [64] shows experiments on other values of d.

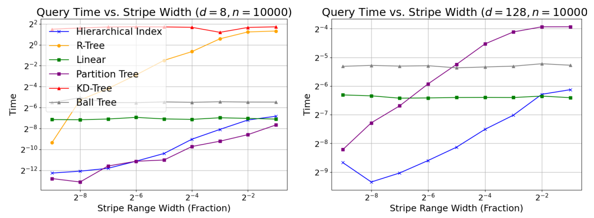

Figure 3: Comparison of SRR query time with respect to the stripe width across different dimensionalities on the synthetic data.

Effect of Dataset Size. Figure 4 illustrates the impact of increasing the dataset size n on the query time of the algorithms, for both low- and high-dimensional settings. As observed, the Hierarchical Sampling method consistently outperforms the baselines with a significant speedup.

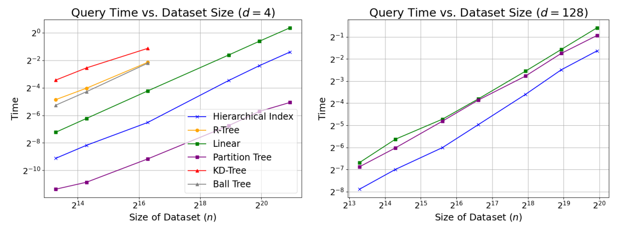

Figure 4: Comparison of SRR query time with respect to dataset size. The higher-dimensional and larger cases do not contain KD-tree and R-tree as these methods do not scale well.

Preprocessing (Indexing) Phase. During the preprocessing phase in all the baselines, an index structure is constructed. Figure 5 (left) shows the memory usage of these indices as a function of the dataset size n, while the right plot illustrates the corresponding preprocessing time. As shown, the space overhead of Hierarchical Sampling is comparable to that of KD-Tree and R-Tree, both of which are commonly used in existing systems. Moreover, the space usage grows linearly with the dataset size n. The time required to build the Hierarchical Sampling index is also on par with that of the Partition Tree and R-Tree.

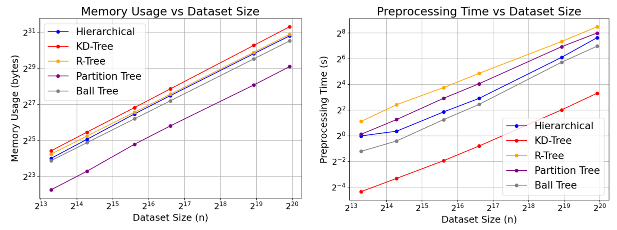

Figure 5: Comparison of SRR indexing time and size with respect to the dataset size for $d = 8$.

Real Datasets. This section shows the performance of methods on real datasets. The results are presented in Figure 6. As shown, Hierarchical Sampling significantly outperforms both Exhaustive search and the Partition Tree method. In this experiment, multiple random scoring functions are sampled uniformly from the unit hypersphere. More experiments on real datasets is presented in Appendix H.2 of the technical report [64].

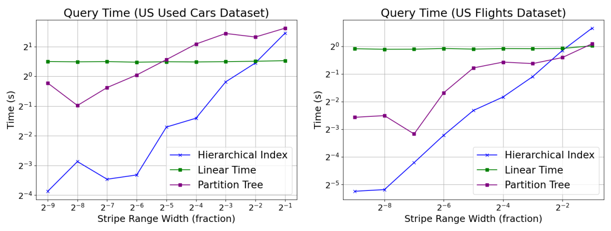

Figure 6: Comparison of SRR query time with respect to the stripe range width. The scoring functions are sampled uniformly at random from a hypersphere.

Accuracy of the Output. In all the settings, the output is guaranteed to have a recall of 100%, reporting all the points in the range.

6.2 Direct-Access and Conformal Set Retrieval

6.2.1 Synthetic Dataset.

Effect of Dimension and value of i. Figure 7 (left) presents the query time of algorithms as a function of dimension d. As d increases, both Fagin and TA algorithms exhibit poor scalability for the DAR queries. Between the proposed methods, EpsHier consistently outperforms EpsRange, particularly when the $d > 8$. The performance degradation of EpsRange in higher dimensions is because of its reliance on Partition Tree index [49], which suffers from the curse of dimensionality.

Figure 7 (right) illustrates the impact of varying the value of i in the DAR query on the runtime. The TA algorithm shows increasing query time as i grows, since it must retrieve all the top-i items. In contrast, the runtime of EpsRange remains relatively stable, as it depends primarily on the position of the stripe rather than the value of i. The EpsHier method also shows a moderate increase in runtime for larger i, which can be because of the relative location of the stripes with respect to the dataset. The marginal stripes (smaller values of i) intersect with fewer balls in the hierarchical structure proposed in Section 5.3, and the pruning is more effective in this case. In contrast, stripes that are relatively in the middle require more balls to be explored.

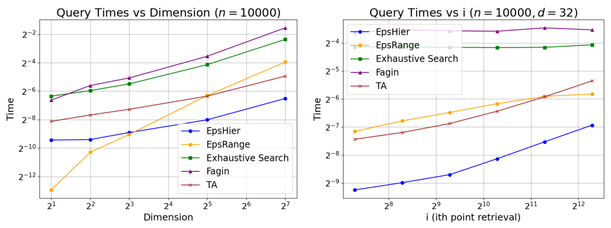

Figure 7: Comparison of CSR query time with respect to the dimension d and the rank i. Here, $\varepsilon = \frac{1}{16}$ for EpsRange and EpsHier.

Effect of Dataset Size and ε. Figure 8 (left) shows the effect of dataset size on the query time of the algorithms. We observe significant speedups achieved by both EpsRange and EpsHier compared to the baseline methods. Figure 8 (right) illustrates the impact of varying the parameter ε. As expected, this parameter influences the size of the output set in the CSR problem. Increasing the value of ε

results in a larger output set, whereas smaller values of ε require searching over a larger ε-sample during the initial phase of the algorithm. Both extremes lead to increased query times. The most efficient performance is typically achieved for a moderate value of ε, which may serve as a default when not given by user.

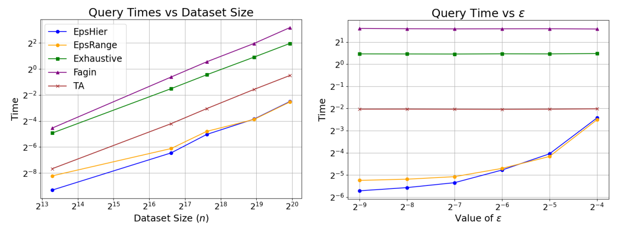

Figure 8: Comparison of query time of CSR with respect to dataset size and ε. In the right plot, $n = 10000$ and the size of returned conformal set is $\varepsilon \cdot n$.

KthLevel Algorithm. Figure 9 shows the result of running the KthLevel algorithm on smaller datasets in 2D. As shown in, the KthLevel algorithm achieves substantial speedups. However, this improvement comes at the cost of increased index size.

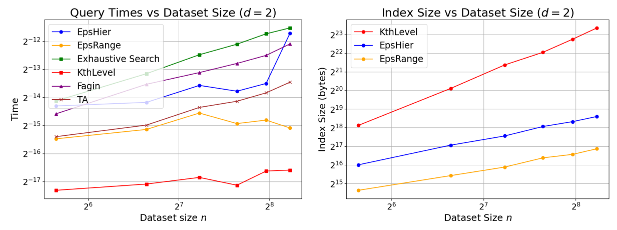

Figure 9: Result of applying KthLevel for solving DAR problem in 2D. Generally, KthLevel does not scale well with dimension d and size n. Here, EpsRange in 2D is equivalent to Eps2D algorithm.

Real Datasets. Figure 10 presents the results of running the algorithms on the US Flights dataset, evaluating performance across different values of ε (left) and different values of i (right). We observe significant speedups achieved by EpsHier, followed by EpsRange, showing their effectiveness in real-world scenarios. Appendix H of the technical report [64] shows the results on other datasets.

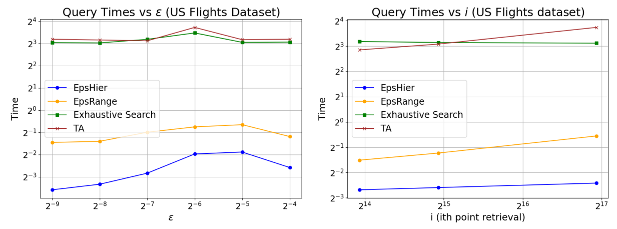

Figure 10: Comparing the CSR query time with respect to value of ε and rank i on US Flights dataset.

Accuracy of the Output. In all the experiments, the returned set is guaranteed to have the $p^{(i)}$, so the recall is always 100%.

7 Related Work

Top-k Retrieval. Top-k query processing has been extensively studied across various settings. The foundational Threshold Algorithm (TA) [33] combines sorted and random access to efficiently identify top-k results and has inspired numerous extensions, including cost-based optimizations and early stopping strategies [14]. Probabilistic top-k methods [65] adapt this problem to uncertain data, ranking results based on expected scores or statistical confidence. Index-based approaches such as the onion layer [17] and view-based techniques [26, 39] have been proposed to accelerate queries through structural or materialized reuse. For comprehensive surveys, see [25, 41]. *In top-k retrieval, k is a small constant*, while in our setting, we aim to find the tuple at rank position i, where i is $O(n)$. As a result, the algorithms proposed for top-k are inefficient for our setting, as shown in the experiments.

Direct Access Queries in Databases. Recent work in database theory has explored the notion of *direct access* to query answers, aiming to support efficient access to the i-th ranked tuple without materializing the full result [15, 31, 66]. Some work extends this to conjunctive queries with aggregation, designing algorithms that enable access to top-ranked answers without fully evaluating the query [31]. Our paper has a different objective. Specifically, rather than assuming a predefined ordering, *we consider a ranking function as part of the user query*. As a result, the proposed algorithms for direct-access queries cannot be adapted to our setting.

Range Searching. Range searching, specifically for simplex ranges, is related to our work in solving the intermediate SRR problem. Some approaches achieve logarithmic query time at the cost of exponential space [4, 29], while others aim for linear space and sublinear query time [21, 38, 48, 49]. Specialized variants have also been studied, including half-space range searching [21] and axis-aligned queries [20]. For a comprehensive survey, see [1]. To address the SRR problem in practice, we develop a hierarchical sampling approach inspired by data structures commonly used for similarity search, like HNSW [47]. As an intermediate problem for solving DAR, *our solutions are tailored for narrow ranges*. While our methods do not offer formal theoretical guarantees, they performed efficiently in practical settings. Another indexing technique related to our algorithm is the *Ball Tree* structure [53]. The key difference between Ball Trees and our approach lies in the construction and structure. In our method, each layer is generated by randomly sampling from the previous one, and each layer serves as a set of centroids for the preceding layer. In contrast, Ball Trees are typically binary trees built in a top-down manner.

Other related topic is *Max Inner Product Search (MIPS)* [7, 62] which focuses on retrieving the item with the highest inner product with a query vector. Techniques, such as asymmetric LSH [62] and tree- or graph-based indices [6, 7], have been proposed to accelerate MIPS. In parallel, *Quantile Sketches* [35, 42] provide compact data summaries that approximate rank-based statistics, enabling efficient quantile estimation in streaming or large-scale settings. Unlike these methods, our work on *Direct Access Retrieval* aims to directly access the ith element according to an arbitrary scoring function.

8 Conclusion

We studied the problem of Direct Access Ranked Retrieval (DAR) and its relaxed variant, Conformal Set Ranked Retrieval (CSR). We first proposed an algorithm based on geometric arrangements with logarithmic query time but an exponential space complexity. Next, we proposed our space-efficient algorithms based on ε-samples. By providing a hierarchical sampling-based data structure for efficient stripe-range retrieval, we developed a practical algorithm for the CSR problem that scales to hundreds of dimensions.

References

[1] Pankaj K Agarwal. 2017. Range searching. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 1057–1092.
[2] Pankaj K Agarwal et al. 1998. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM journal on computing* 27, 3 (1998), 654–667.
[3] Pankaj K Agarwal, Ravid Cohen, Dan Halperin, and Wolfgang Mulzer. 2019. Dynamic maintenance of the lower envelope of pseudo-lines. *arXiv preprint arXiv:1902.09565* (2019).
[4] Alok Aggarwal, Mark Hansen, and Thomas Leighton. 1990. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 331–340.
[5] Anastasios N Angelopoulos and Stephen Bates. 2021. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511* (2021).
[6] Alex Auvolat et al. 2015. Clustering is efficient for approximate maximum inner product search. *arXiv preprint arXiv:1507.05910* (2015).
[7] Yoram Bachrach et al. 2014. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*. 257–264.
[8] R Ricardo Baeza-Yates et al. 1999. *Modern information retrieval*. ACM Press.
[9] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
[10] Nikos Bikakis, Panos K Chrysanthis, Guoliang Li, George Papastefanatos, and Lingyun Yu. 2025. Visual Analytics Challenges and Trends in the Age of AI: The BigVis Community Perspective. *ACM SIGMOD Record* 54, 2 (2025), 66–69.
[11] Manuel Blum et al. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461.
[12] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1-7 (1998), 107–117.
[13] Andrei Broder. 2002. A taxonomy of web search. In *ACM Sigir forum*, Vol. 36. ACM New York, NY, USA, 3–10.
[14] N. Bruno, L. Gravano, and A. Marian. 2002. Evaluating top-k queries over Web-accessible databases. In *IEEE ICDE*. 369–380.
[15] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. 2023. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Transactions on Database Systems* 48, 1 (2023), 1–45.
[16] Timothy M Chan. 2020. Dynamic geometric data structures via shallow cuttings. *Discrete & Computational Geometry* 64, 4 (2020), 1235–1252.
[17] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith. 2000. The onion technique: indexing for linear optimization queries. *ACM SIGMOD* 29, 2 (2000), 391–402.
[18] Bernard Chazelle. 1989. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society* 2, 4 (1989), 637–666.
[19] Bernard Chazelle. 2000. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press.
[20] Bernard Chazelle and Leonidas J Guibas. 1986. Fractional cascading: I. A data structuring technique. *Algorithmica* 1, 1 (1986), 133–162.
[21] Bernard Chazelle, Leo J Guibas, and Der-Tsai Lee. 1985. The power of geometric duality. *BIT Numerical Mathematics* 25, 1 (1985), 76–90.
[22] Fanchao Chen, Dixin Tang, Haotian Li, and Aditya G Parameswaran. 2023. Visualizing Spreadsheet Formula Graphs Compactly. *Proceedings of the VLDB Endowment* 16, 12 (2023), 4030–4033.
[23] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.
[24] W Bruce Croft, Donald Metzler, Trevor Strohman, et al. 2010. *Search engines: Information retrieval in practice*. Vol. 520. Addison-Wesley Reading.
[25] Gautam Das. 2009. Top-k Algorithms and Applications. In *DASFAA*. 789–792.
[26] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. 2006. Answering top-k queries using views. In *VLDB*. 451–462.
[27] Mohsen Dehghankar and Abolfazl Asudeh. 2025. HENN: A Hierarchical Epsilon Net Navigation Graph for Approximate Nearest Neighbor Search. *CoRR, abs/2505.17368* (2025).
[28] Tamal K Dey. 1998. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry* 19, 3 (1998), 373–382.
[29] David Dobkin, John Hershberger, David Kirkpatrick, and Subhash Suri. 1990. Implicitly searching convolutions and computing depth of collision. In *International Symposium on Algorithms*. Springer, 165–180.
[30] Herbert Edelsbrunner. 1987. *Algorithms in combinatorial geometry*. Vol. 10. Springer Science & Business Media.
[31] Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. 2023. Direct access for answers to conjunctive queries with aggregation. *arXiv preprint arXiv:2303.05327* (2023).
[32] Hazel Everett et al. 1993. An optimal algorithm for the ($\leq$ k)-levels, with applications to separation and transversal problems. In *Proceedings of the ninth annual symposium on Computational geometry*. 38–46.
[33] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *ACM PODS*. 102–113.
[34] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. 2002. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems (TODS)* 27, 3 (2002), 261–298.
[35] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record* 30, 2 (2001), 58–66.
[36] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD*. 47–57.
[37] Sariel Har-Peled. 2011. *Geometric approximation algorithms*. Number 173. American Mathematical Soc.
[38] David Haussler and Emo Welzl. 1986. Epsilon-nets and simplex range queries. In *Proceedings of the second annual symposium on Computational geometry*. 61–71.
[39] Vagelis Hristidis et al. 2004. Algorithms and applications for answering ranked queries using ranked views. *The VLDB Journal* 13, 1 (2004), 49–70.
[40] Ihab F Ilyas et al. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* 40, 4 (2008), 1–58.
[41] Ihab F. Ilyas et al. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surveys* 40, 4, Article 11 (2008).
[42] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.
[43] Thomas Kluyver et al. 2016. Jupyter Notebooks–a publishing format for reproducible computational workflows. In *Positioning and power in academic publishing: Players, agents and agendas*. IOS press, 87–90.
[44] Stefano Leone. 2022. FIFA 23 Player Dataset. https://www.kaggle.com/datasets/stefanoleone992/fifa-23-complete-player-dataset. Accessed: 2025-07-31.
[45] Huajing Li, Yuan Tian, Wang-Chien Lee, C Lee Giles, and Meng-Chang Chen. 2010. Personalized feed recommendation service for social networks. In *2010 IEEE Second International Conference on Social Computing*. IEEE, 96–103.
[46] Zeyao Ma et al. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems* 37 (2024), 94871–94908.
[47] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
[48] Jiří Matoušek. 1991. Efficient partition trees. In *Proceedings of the seventh annual symposium on Computational geometry*. 1–9.
[49] Jiri Matousek. 1992. Reporting points in halfspaces. *Computational Geometry* 2, 3 (1992), 169–186.
[50] Julian McAuley et al. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 43–52.
[51] Stefan Meiser. 1993. Point location in arrangements of hyperplanes. *Information and Computation* 106, 2 (1993), 286–303.
[52] Ananay Mital. 2020. US Used Cars Dataset. https://www.kaggle.com/datasets/ananaymital/us-used-cars-dataset. Accessed: 2025-07-31.
[53] Stephen M Omohundro. 1989. Five balltree construction algorithms. (1989).
[54] Stephen M Omohundro. 1989. Five balltree construction algorithms. (1989).
[55] Serhii Orlivskyi, Bohdan Deomin, and Olga Averianova. 2021. Pagination And Its Efficient Methods For RESTful Web Services. In *2021 IEEE 3rd Ukraine Conference on Electrical and Computer Engineering (UKRCON)*. IEEE, 567–571.
[56] Sajjadur Rahman et al. 2020. Benchmarking spreadsheet systems. In *Proceedings of the 2020 ACM SIGMOD*. 1589–1599.
[57] Carl M Rebman Jr et al. 2023. An Industry Survey of Analytics Spreadsheet Tools Adoption: Microsoft Excel vs Google Sheets. *Information Systems Education Journal* 21, 5 (2023), 29–42.
[58] Francesco Ricci et al. 2021. Recommender systems: Techniques, applications, and challenges. *Recommender systems handbook* (2021), 1–35.
[59] Glenn Shafer and Vladimir Vovk. 2008. A tutorial on conformal prediction. *Journal of Machine Learning Research* 9, 3 (2008).
[60] Shreya Shankar, Stephen Macke, Sarah Chasins, Andrew Head, and Aditya Parameswaran. 2022. Bolt-on, compact, and rapid program slicing for notebooks. *Proceedings of the VLDB Endowment* 15, 13 (2022), 4038–4047.
[61] Micha Sharir, Shakhar Smorodinsky, and Gábor Tardos. 2000. An improved bound for k-sets in three dimensions. In *Proceedings of the sixteenth annual symposium on Computational geometry*. 43–49.
[62] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems* 27 (2014).
[63] Parsharam Reddy Sudda. 2024. Optimizing PHP API calls with pagination and caching. (2024).
[64] Technical Report Anonymized for double-blind review [n.d.]. https://anonymous.4open.science/r/DAR-F300/technical_report.pdf Anonymized Link.
[65] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. 2004. Top-k query evaluation with probabilistic guarantees. In *VLDB*. 648–659.
[66] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2024. Ranked enumeration for database queries. *ACM SIGMOD Record* 53, 3 (2024), 6–19.
[67] Bureau of Transportation Statistics U.S. Department of Transportation. n.d.. Flight Delays and Cancellations. https://www.kaggle.com/datasets/usdot/flight-delays?select=flights.csv. Accessed: 2025-10-04.

[68] Vladimir N Vapnik and A Ya Chervonenkis. 2015. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity: festschrift for alexey chervonenkis*. Springer, 11–30.
[69] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.
[70] Emo Welzl. 2005. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings*. Springer, 359–370.
[71] Jinjin Zhao et al. 2023. Data Makes Better Data Scientists. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–3.

Appendix

A Algorithm Based on k-th Level of Arrangement

In this section, we describe the details of KthLevel algorithm. We begin by introducing key concepts from computational geometry, and then describe our algorithm in detail.

A.1 Preliminaries

Duality. We use the classic dual space transformation that maps points to hyperplanes and vice versa. This transformation is well-defined and preserves intersection relationships [21, 30]. Formally, the dual of a point $p \in \mathbb{R}^d$ is defined as:

$$\mathsf{dual}(p) := \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^\top p = 1 \right\}.$$

In other words, the transformation $\mathsf{dual}(\cdot)$ maps a point in the *primal space* to a hyperplane in the *dual space*. This hyperplane is represented by the equation:

$$h : \sum_{i=1}^{d} p[i] \cdot x_i = 1.$$

We refer to the original input domain as the *primal space*, and the space of dual hyperplanes as the *dual space*. Figure 11 provides a visual illustration of this transformation. The left figure shows the primal space, while the middle figure shows the dual hyperplanes in the dual space. The following key observation follows from this duality. A formal proof can be found in [30].

Lemma 6. *Let f be a scoring function (i.e., a direction vector), and let $p^{(1)}, \dots, p^{(n)}$ be the points sorted in descending order according to their score under f, so that $f^\top p^{(1)} > f^\top p^{(2)} > \cdots$. Then, for any $a < b$, the intersection point of $\mathsf{dual}(p^{(b)})$ with the ray in direction f lies closer to the origin than that of $\mathsf{dual}(p^{(a)})$.* □

For instance, in Figure 11, the point B has the highest score under f_1, and thus its dual $\mathsf{dual}(B)$ intersects the ray f_1 in the dual space earlier (closer to origin) than any other point's dual.

A.1.1 k-th Level of Arrangement. Consider the set of dual hyperplanes $\mathcal{H} = \{\mathsf{dual}(p) \mid p \in \mathcal{D}\}$. $\mathcal{H}$ defines a dissection of $\mathbb{R}^d$ into connected convex cells, called the *arrangement* of the hyperplanes [30].

To simplify the explanations, let $d = 2$. Sweep a ray f counterclockwise from the x-axis to the y-axis. At each orientation of f, the ray intersects all dual hyperplanes (dual lines) in $\mathcal{H}$. These intersection points can be sorted by their distance from the origin. The *k-th level* is the locus of points that lie on the k-th closest dual line intersected by f, as the direction of f varies continuously [30].

In general d-dimensional space, consider a ray f originating at the origin and sweeping over all directions on the unit sphere $\mathbb{S}^{d-1}$. For each direction f, the ray intersects the hyperplanes in $\mathcal{H}$ in up to n distinct points, which can again be ordered by increasing distance from the origin. The *k-th level* is the set of all such k-th intersection points across all directions $f \in \mathbb{S}^{d-1}$.

Figure 11 illustrates the first and second levels of the arrangement in $\mathbb{R}^2$, shown in orange and green, respectively.

A.2 KthLevel Algorithm

In this subsection, we formally present our algorithm, KthLevel. We begin by describing the *preprocessing* phase, followed by a discussion of the *query phase*, and conclude with a theoretical analysis of the algorithm's performance. The overall structure of the algorithm applies uniformly across dimensions $d \geq 2$; however, for the purpose of analysis, we distinguish between the planar case ($d = 2$) and the general higher-dimensional case ($d > 2$).

Preprocessing. During the preprocessing phase, we construct the arrangement of the dual hyperplanes induced by the point set $\mathcal{D}$, and the levels of the arrangement for all $1 \leq k \leq n$. We assume access to an oracle $\mathcal{K}$ that, for a given i, returns an efficient data structure representing the i-th level of the arrangement [30]. By calling this oracle for each $i \in [n]$, we obtain the complete set of structures needed for query processing. Each such structure allows constant-time access to the hyperplanes intersecting on each face of the arrangement, as well as the points in the primal space corresponding to those hyperplanes [30].

Query Phase. In the query phase, we are given a scoring function f and a rank i, and the goal is to retrieve the point $p^{(i)}$, which is the i-th highest-ranked point with respect to f. By Lemma 6, the intersection of the i-th level of arrangement of dual hyper-planes with f corresponds with the desired point $p^{(i)}$. Algorithm 3 shows the pseudo-code of our process for finding this intersection.

Consider the 3rd level of the arrangement highlighted (in orange) in the Figure 11 (right figure). In 2D, the k-th level is a non-convex chain of line segments, each belonging to a dual line. Now, let the query function be f_2, while $i = 3$. Our objective is to find $p^{(3)} = \mathsf{rank}_{f_2}^{-1}(3)$. The line segment in the 3rd level that intersects with the vector of f_2 is $\mathsf{dual}(D)$. Hence, the point $p^{(3)} = D$ in this example. Note that applying a binary search on the endpoints of the line segments of the 3rd level, one can find the intersection of any function f with any level i in a time logarithmic to the number of line segments.

From the above example, observe that the k-th level is a chain of line segments. Similarly, for a general $d \geq 2$, the k-th level is a chain of $(d-1)$-faces.[10] Each function f, in such cases, is an origin-anchored vector that intersects one of the faces of each level $i, \forall i \in [n]$. Projecting the endpoints of these faces on the surface of the unit sphere $\mathbb{S}^{d-1}$ forms an arrangement in the $(d-1)$-dimensional space. For example, when $d = 3$, each k-th level is a chain of plane segments, and their projection onto the unit ball (centered at the origin) is an arrangement of lines.

Now, consider the intersection point of the query function f with the unit sphere $\mathbb{S}^{d-1}$. Finding the face segment of a level i with a function f is equivalent to finding the cell of the arrangement on

[10] A d'-face of an arrangement is a d'-dimensional convex cell, formed by the intersection of $d - d'$ hyperplanes. For example, 0-faces are vertices (points), 1-faces are edges (segments or lines), and so on.

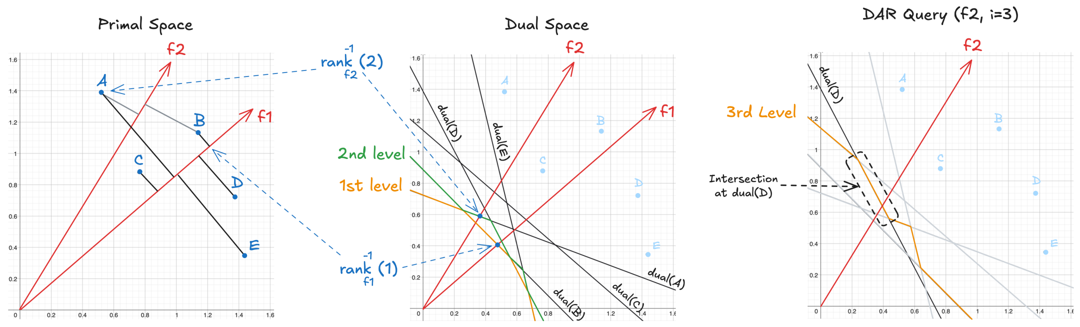

Figure 11: A 2D representation of dual space and the k-th level of arrangements (For simplicity, only the first quadrant is presented). The right figure shows an example of running KthLevel for $i = 3$. The result of this query is the point D.

Algorithm 3 KthLevel-Query(f, i)

```
Input: The scoring function f and the order i
Output: The i-th highest scored point according to f: $p^{(i)}$
1: $Level_i \leftarrow$ preprocessed data structure        ▷ Get the i-th level
2: $face \leftarrow$ Intersection(f, Level_i)               ▷ Get the intersection
3: $h \leftarrow$ The hyperplane corresponding to $face$ in arrangement
4: $p^{(i)} \leftarrow \text{dual}^{-1}(h)$
5: return $p^{(i)}$
```

the surface of the sphere that intersects with f. This problem is known as *point location* on the arrangement of hyperplanes [51]. Line 2 of Algorithm 3 uses a state-of-art point-location algorithm for finding the intersection of f with the i-th level.

A.3 Analysis of KthLevel Algorithm

Preprocessing. In 2D, the complexity of the k-th level in an arrangement of n lines in the plane is $O(nk^{1/3})$ [28]. Hence, considering an arbitrary level $i = O(n)$, the complexity of the i-th level is $O(n^{4/3})$. The total preprocessing time to construct all levels of the arrangement is $O(n^2)$, where each individual level can be built in $O(n \log n + nk)$ time [32]. In 3D, the complexity of the k-th level is $O(nk^{3/2})$ [61]. As a result, the complexity of each level $i \in [n]$ is bounded by $O(n^{5/2})$. The complexity of the arrangement in the general d-dimensional setting is $O(n^d)$, and the total time required to construct it is also $O(n^d)$ [2, 30].

Query Time. During the query time, Algorithm 3 finds the intersection of the i-th level with the vector of the function f. Given that the total number of faces in each level $i \in [n]$ is bounded by $O(n^d)$, identifying this intersection using a state-of-the-art algorithm for point location on the arrangement of hyperplanes is in $O(\log n)$, ignoring the constant factors that depend on d [51].

B Missing Proofs

B.1 Proof of Lemma 2

Proof. Let $\mathcal{H}$ be the half-space defined by: $\mathcal{H} = \{x \in \mathbb{R}^d \mid f^\top x \geq f^\top p^{(i)}\}$. This half-space is perpendicular to the direction vector f and contains exactly the top i points of $\mathcal{D}$ ranked by f, including $p^{(i)}$. In other words, $|\mathcal{H} \cap \mathcal{D}| = i$.

Since $\mathcal{N}_\varepsilon$ is an ε-sample of $\mathcal{D}$ for half-space (and slab) ranges, it preserves the proportion of points in any half-space up to additive error ε. Therefore,

$$\left| \frac{|\mathcal{H} \cap \mathcal{N}_\varepsilon|}{|\mathcal{N}_\varepsilon|} - \frac{|\mathcal{H} \cap \mathcal{D}|}{|\mathcal{D}|} \right| \leq \varepsilon.$$

Substituting $|\mathcal{H} \cap \mathcal{D}| = i$, and denoting $n = |\mathcal{D}|$ and $m = |\mathcal{N}_\varepsilon|$, we obtain:

$$\left| \frac{|\mathcal{H} \cap \mathcal{N}_\varepsilon|}{m} - \frac{i}{n} \right| \leq \varepsilon,$$

which implies:

$$m\left(\frac{i}{n} - \varepsilon\right) \leq |\mathcal{H} \cap \mathcal{N}_\varepsilon| \leq m\left(\frac{i}{n} + \varepsilon\right).$$

Rounding to integers, define:

$$i_\ell = \left\lfloor m\left(\frac{i}{n} - \varepsilon\right)\right\rfloor, \quad i_u = \left\lceil m\left(\frac{i}{n} + \varepsilon\right)\right\rceil.$$

These bounds indicate that the number of points in $\mathcal{N}_\varepsilon$ with score at least $\texttt{score}_f(p^{(i)})$ lies between i_ℓ and i_u. Equivalently, the score $\texttt{score}_f(p^{(i)})$ lies between the scores of the i_u-th and i_ℓ-th ranked points in $\mathcal{N}_\varepsilon$. We conclude that:

$$\texttt{score}_f(q_\ell) \leq \texttt{score}_f(p^{(i)}) \leq \texttt{score}_f(q_u),$$

as desired. □

B.2 Proof of Lemma 3

Proof. The result set $\mathcal{D}_o$ consists of the points in the dataset $\mathcal{D}$ that lie between the i_ℓ-th and i_u-th ranked points in the sample $\mathcal{N}_\varepsilon$. Define the corresponding score interval as the stripe range:

$$S = \{x \in \mathbb{R}^d \mid f^\top q_\ell \leq f^\top x \leq f^\top q_u\}.$$

Then, $|\mathcal{D}_o| = |S \cap \mathcal{D}|$. Since $\mathcal{N}_\varepsilon$ is an ε-sample of $\mathcal{D}$ for stripe ranges, we have (with high probability): $\left|\frac{|S \cap \mathcal{N}_\varepsilon|}{|\mathcal{N}_\varepsilon|} - \frac{|\mathcal{D}_o|}{|\mathcal{D}|}\right| \leq \varepsilon$. From Equation 1, we know that $|S \cap \mathcal{N}_\varepsilon| = |i_u - i_\ell| \leq 2m\varepsilon + 2$. Hence,

$$\left|\frac{|S \cap \mathcal{N}_\varepsilon|}{m} - \frac{|\mathcal{D}_o|}{n}\right| \leq \varepsilon \Longrightarrow \frac{|\mathcal{D}_o|}{n} \leq \frac{2m\varepsilon + 2}{m} + \varepsilon \Longrightarrow |\mathcal{D}_o| = O(\varepsilon n).$$

□

C Dynamic Setting

In this section, we discuss the dynamic setting. We consider two types of dynamic operations: `Insert(p)` and `Delete(p)`. For the KthLevel algorithm, these operations are supported following the literature on dynamically maintaining levels of arrangements [3, 16]. Below, we focus on the EpsHier algorithm; analogous results can be shown for EpsRange.

`Insert(p)`. This operation adds a new point p to the dataset $\mathcal{D}$. For EpsHier, we must update both the ε-sample $\mathcal{N}_\varepsilon$ and the hierarchical sampling structure described in Section 5.3. According to Theorem 1, any random subset of $\mathcal{D}$ of size $|\mathcal{N}_\varepsilon|$ is an ε-sample. Therefore, our goal is to maintain such a random subset without resampling the entire dataset. This can be efficiently achieved using *reservoir sampling* [69], which allows us to maintain a uniform random sample as new elements arrive. To update the hierarchical structure used for the SRR problem, we apply this reservoir sampling procedure bottom-up, from the base layer to higher layers. At each layer, we decide whether to include the new point, as each layer is a random sample of the one below it. This process takes $O(\log n)$ time since the index contains $\log n$ layers.

`Delete(p)`. This operation removes p from $\mathcal{D}$. Under the *backing sample* method [34], if $p \in \mathcal{N}_\varepsilon$, we delete it from the sample. If the sample size drops below a predefined lower bound L, we *rebuild* $\mathcal{N}_\varepsilon$ by resampling all the points. The amortized time for this operation (based on the definition of L) is $O(1)$ [34]. For the hierarchical structure, each layer ℓ is maintained similarly, with its own bound L_ℓ, and each layer ℓ is the sample maintained at level $\ell-1$. Hence, a rebuild at level ℓ only scans the immediately lower level, not the entire dataset, keeping the amortized cost low. In other words, this rescanning at layer ℓ takes $O(|\mathcal{L}_{\ell-1}|) = O(\frac{n}{2^{\ell-1}})$, but the amortized is still $O(1)$ for this layer and it takes $O(\log n)$ to resample all the top layers. As a result, the total amortized time for this operation is $O(\log n)$ for the hierarchical structure.

D Analysis of EpsRange Algorithm

The analysis is similar to Eps2D, however, the time complexity of the algorithm depends on the specific range counting and simplex range searching we use at steps ii and iii.

Step i takes $O(m)$ for finding the thresholds in the ε-sample. Step ii requires solving the SRR problem. Following the simplex range searching algorithms, we can solve this problem in $O(n^{1-1/d}+|\mathcal{D}_o|)$ with $O(n)$ space [49]. To report the exact response to DAR, i.e., $p^{(i)}$, we proceed to steps iii and iv. In step iii, we solve an instance of range counting in d dimensions, which takes $O(n^{1-1/d})$ [49] time and linear space. Finally, in step iv, we sort $\mathcal{D}_o$, which takes $O(|\mathcal{D}_o|\log|\mathcal{D}_o|)$.

Space Complexity. The space complexity of this algorithm is linear to the input size, since all structures require linear space.

Time Complexity for the CSR problem. The total query time of the algorithm for finding the conformal set is:

$$T(\text{EpsRange}_{\text{CSR}}) = O(m + n^{1-1/d})$$

The size of ε-sample m is $O(\frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon})$, since the VC-dimension of stripe range is $\delta = d+1$ [37, 38]. This gives us the following query time:

$$T(\text{EpsRange}_{\text{CSR}}) = O(\max\{n^{1-1/d}, \frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}\}) = \Omega(n^{1-1/d}).$$

The output is a set $\mathcal{D}_o$ with size $O(\varepsilon n)$, guaranteed to contain $p^{(i)}$ (see Lemma 3). Note that the final time also depends on the choice of parameter ε, which determines the size of output $\mathcal{D}_o$.

Time Complexity for the DAR problem. The exact version also includes the steps iii and iv:

$$T(\text{EpsRange}_{Exact}) = O(m + n^{1-1/d} + |\mathcal{D}_o|\log|\mathcal{D}_o|).$$

Based on Lemma 3, we have $|\mathcal{D}_o| = O(\varepsilon n)$. As a result, by choosing $\varepsilon = n^{-1/3}$, we get the following runtime:

$$T(\text{EpsRange}_{Exact}) = O(\max\{n^{1-1/d}, dn^{2/3}\log n\})$$

Theorem 7. *The EpsRange algorithm solves the relaxed version of DAR problem with linear space usage and $O(\max\{n^{1-1/d}, \frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}\})$ time. The exact version can also be solved with the same space usage and $O(\max\{n^{1-1/d}, dn^{2/3}\log n\})$ time.*

E Analysis of Hierarchical Sampling

The total space usage for storing the hierarchical layers is linear to the input, since the size of each layer decreases exponentially:

$$\sum_{\ell\le L}|\mathcal{L}_\ell| = \sum_{\ell\le L}\frac{n}{r^\ell} = O(n),$$

assuming $r = O(1)$. The runtime of Algorithm 4 is dominated by the **for** loop at line 6, where the layers are constructed, and edges and area sets are updated. At each layer ℓ, we sample $|\mathcal{L}_\ell|$ points. Then, for each point in $\mathcal{L}_{\ell-1}$, we compute its nearest neighbor in $\mathcal{L}_\ell$. The time complexity at the layer ℓ is:

$$T_\ell = O\left(d\cdot|\mathcal{L}_\ell|\cdot|\mathcal{L}_{\ell-1}|\right).$$

Summing over all layers, the total runtime becomes:

$$\sum_\ell T_\ell = \sum_\ell d\cdot\frac{n}{r^\ell}\cdot\frac{n}{r^{\ell-1}} = \sum_\ell\frac{dn^2}{r^{2\ell-1}} = O(dn^2),$$

again assuming $r = O(1)$. For computing the smallest enclosing ball of each area, we use Welzl's algorithm, which runs in linear time with respect to the number of points [70].

Theorem 8. *The preprocessing phase of the hierarchical sampling algorithm takes $O(dn^2)$ time and uses linear space.*

F Additional Pseudo-codes

Algorithm 4 Hierarchical-Sampling-Preprocess($\mathcal{D}, r$)

```
Input: The dataset $\mathcal{D}$ and the exponential decay rate $r$
Output: The hierarchical graph $\mathcal{G}(\mathcal{D})$ which contains a list of
    layers $\mathcal{L}$, neighboring relations $N$, and enclosing balls of nodes
    $\mathcal{B}$.
 1: $L \leftarrow \lfloor\log_r n\rfloor$                          ▷ Number of layers
 2: $\mathcal{L}_\ell \leftarrow [\,]$,   $\forall \ell \le L$          ▷ Placeholder for the layers
 3: $\mathcal{L}_0 \leftarrow \mathcal{D}$                                  ▷ Base layer
 4: $\mathcal{A}_0(p) = \{p\}$,   $\forall p \in \mathcal{L}_0$                  ▷ Area of nodes
 5: $N_0(p) \leftarrow \varnothing$,   $\forall p \in \mathcal{L}_0$                       ▷ Neighbors
 6: for layer $\ell = 1$ to $L$ do
 7:     $\mathcal{L}_\ell \leftarrow$ Random sample of size $\frac{|\mathcal{L}_{\ell-1}|}{r}$ from $\mathcal{L}_{\ell-1}$
 8:     $\mathcal{A}_\ell(p) \leftarrow \varnothing$,   $\forall p \in \mathcal{L}_\ell$
 9:     $N_\ell(p) \leftarrow \varnothing$,   $\forall p \in \mathcal{L}_\ell$
10:     for point $p \in \mathcal{L}_{\ell-1}$ do
11:         $c^* \leftarrow \arg\min_{c\in\mathcal{L}_\ell} Dist(p,c)$            ▷ Equation 4
12:         $N_\ell(c^*) \leftarrow N_\ell(c^*) \cup \{p\}$                ▷ Add neighbor
13:         $\mathcal{A}_\ell(c^*) \leftarrow \mathcal{A}_\ell(c^*) \cup \mathcal{A}_{\ell-1}(p)$            ▷ Update area
14: $\mathcal{B}_\ell(p) \leftarrow$ EnclosedBall($\mathcal{A}_\ell(p)$),   $\forall \ell \le L, p \in \mathcal{L}_\ell$
15: $\mathcal{L} \leftarrow \{\mathcal{L}_\ell \mid \ell \le L\}$
16: $N \leftarrow \{N_\ell(p) \mid \forall \ell \le L, p \in \mathcal{L}_\ell\}$
17: $\mathcal{B} \leftarrow \{\mathcal{B}_\ell(p) \mid \forall \ell \le L, p \in \mathcal{L}_\ell\}$
18: return $(\mathcal{L}, N, \mathcal{B})$          ▷ layers, edges, and enclosing balls
```

G Lower Bounds and Optimality of Our Algorithms

In this section, we analyze the complexity of the DAR problem and show the near-optimality of our proposed algorithms. Specifically, the well-studied *half-space range counting* problem [1, 18] reduces to our problem. Hence, the complexity of the state-of-the-art algorithms for half-space range counting provides a lower bound for the DAR problem.

The half-space range counting problem is defined as follows: given a point set in $\mathbb{R}^d$ and a query half-space H, determine the number of points lying within H.

LEMMA 9. *Let $\mathcal{A}$ be an algorithm that solves the DAR problem with query time $T(\mathcal{A})$ and space complexity $S(\mathcal{A})$. Then, using $\mathcal{A}$, one can solve an instance of the half-space range counting problem using space $S(\mathcal{A})$ and query time $\Theta(\log n \cdot T(\mathcal{A}))$.*

PROOF. To count the number of points within a given half-space H, we can perform a binary search over $k \in \{1, 2, \ldots, n\}$. In each iteration, we query $\mathcal{A}$ to retrieve the k-th ranked point $p^{(k)}$. If $p^{(k)} \in H$, then the count of points in H is at least k; otherwise, it is less. This process requires $\Theta(\log n)$ queries to $\mathcal{A}$, resulting in total query time $\Theta(\log n \cdot T(\mathcal{A}))$. □

It is known that the query time for half-space range counting in $\mathbb{R}^d$ with m units of space satisfies the following lower bound [18]:

$$\Omega\left(\frac{n}{m^{1/d}\log n}\right). \tag{5}$$

Applying Lemma 9, this implies a lower bound for the DAR problem:

$$T(\mathcal{A}) \cdot \log n \geq \frac{n}{m^{1/d}\log n} \implies T(\mathcal{A}) \geq \frac{n}{m^{1/d}\log^2 n}.$$

Now, these are the two important observations:

Observation 1. If the algorithm uses linear space, i.e., $m = O(n)$, then the query time cannot be faster than $O(n^{1-\frac{1}{d}})$ up to logarithmic factors.

Observation 2. If the algorithm uses exponential space, i.e., $m = 2^{\Theta(d)}$, then the lower bound becomes $\Omega(\log n)$. In fact, our exact algorithm KTHLEVEL has logarithmic query time but requires exponential space, which matches this bound. Note that querying KTHLEVEL n times is equivalent to sorting all n elements, which takes $\Omega(n \log n)$ time.

Conclusion. These results demonstrate that our algorithms for the DAR problem are optimal up to logarithmic factors, given the space-time trade-offs imposed by known lower bounds.

H More on Experiments

H.1 Dataset Details

We use both synthetic and real-world datasets for our evaluation. The synthetic datasets are generated using the Zipfian distribution, with variations in dimensionality, dataset size, and distribution parameters. The Zipfian distribution is particularly challenging for range searching due to its inherent skewness and is commonly observed in many real-world scenarios. It allows us to evaluate the algorithms under non-uniform workloads.

For our real-world experiments, we use three datasets: the US Used Cars dataset [52], the FIFA 2023 dataset [44], and the US Flights dataset [67]. The US Used Cars dataset contains approximately 3 million entries with 66 attributes describing vehicle specifications, market prices, and regional features, making it suitable for evaluating ranking and filtering performance over high-dimensional numeric attributes. The FIFA 2023 dataset comprises around 300,000 records with 54 player and team attributes, including overall ratings, skills, and physical characteristics, and provides a structured benchmark for similarity and top-k retrieval tasks. Finally, the US Flights dataset contains over 5 million flight records with 31 attributes collected from the U.S. Department of Transportation, including flight times, delays, cancellations, and carrier information, offering a real-world testbed with temporal and categorical dimensions for large-scale ranked retrieval evaluation.

Scoring Functions. In our experiments, we generate a uniformly random unit vector f as the ranking function on all the above datasets.

H.2 More on Stripe Range Retrieval

Figure 12 shows the query time of different methods, when value of d is 4 and 32.

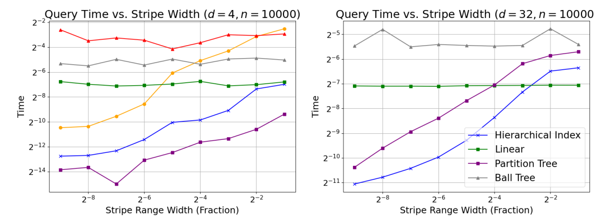

Figure 12: Comparison of SRR query time with respect to the stripe width across different dimensionalities on the synthetic data.

Figure 13 shows the experiments on FIFA dataset.

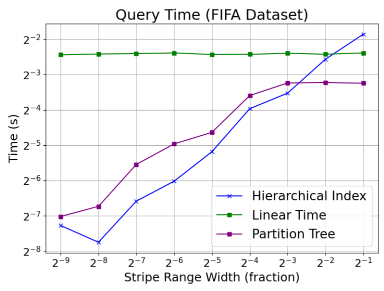

Figure 13: Comparison of SRR query time with respect to the stripe range width. The scoring functions are sampled uniformly at random from a hypersphere.

H.3 More on Direct-Access and Conformal Set Retrieval

Figures 14 and 15 compare the performance of the algorithms on the US Used Cars and FIFA datasets, where the parameters ε and rank i are varied across different runs.

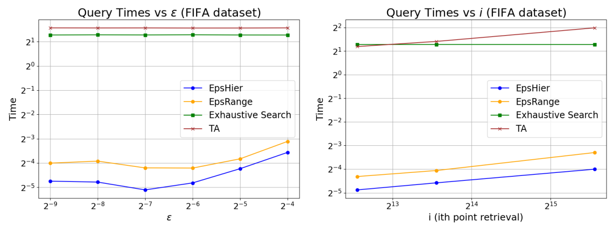

Figure 14: Comparing the CSR query time with respect to value of ε and rank i on FIFA 2023 dataset.

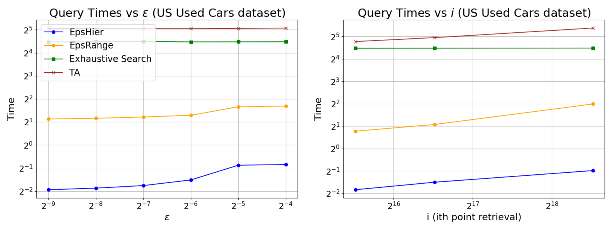

Figure 15: Comparing the CSR query time with respect to value of ε and rank i on US Used Cars Dataset.